Connect 4, Reinforcement Learning

Tristan Basler

CentraleSupélec

Avec

Clément Boulay

Nicolas Noblot

https://github.com/bclement1/connect-4-agent

Abstract

Depuis 2017 et l'arrivée d'AlphaGo dans le monde des échecs et du Go, l'intérêt autour de l'apprentissage par renforcement n'a cessé de croître. De plus, l'entreprise DeepMind ne s'est pas arrêté à ce succès. En effet, en 2018, elle organise des parties de StarCraft2, jeu de stratégie en temps réel, contre des joueurs professionnels. Par ces deux succès notables, DeepMind a permis d'amener l'apprentissage par renforcement sur le devant de la scène de l'intelligence artificielle. Nous allons donc nous aussi comment l'apprentissage par renforcement permet de construire des agents, imbattables dans les jeux.

1. Introduction

1.1. Problème étudiée :

Nous avons choisi d'étudier le jeu du puissance 4. Dans une grille de 7 colonnes de tailles 6, le joueur doit aligner en ligne, en colonne ou en diagonale 4 pions de sa couleur. Contrairement au morpion, la gravité est prise en compte dans ce jeu. Ainsi, le joueur ne peut jouer que dans la case la plus basse d'une colonne.

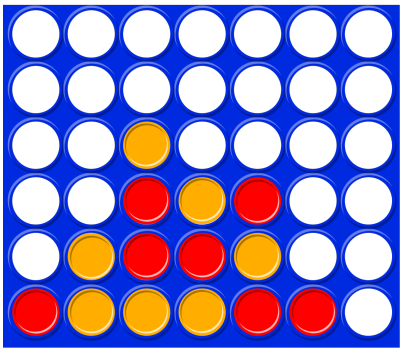

Figure 1. Exemple de parties de puissance 4

1.2. Approche proposée :

Bien que le puissance 4 soit résolue de manière exacte depuis 1988 [1], son étude sous l'angle d'apprentissage par renforcement reste intéressante. En effet, son environnement permet d'appliquer de notre méthode, comme des réseaux de neurones, des chaînes de markov etc. De plus, son espace de possibilités, environ 10^{20} est dans un entre-deux entre espaces petits, comme le morpion (environ 20 000 positions possibles), et espaces grands, comme les échecs (plus de 10^{120} possibilités) ou le go.

Ainsi, nous proposons de construire différents types d'agents, basés sur différentes méthodes d'apprentissages par renforcement, plutôt que de construire un programme formel basé sur la résolution du jeu. Cependant, au vue du fait que le premier joueur est sûr de gagner, il nous semble nécessaire d'alterner le premier joueur à chaque manche.

2. Acteur proposé

2.1. Acteur :

D'un manière similaire qu'AlphaGo [2], nous avons implémenté un agent basé sur l'évaluation des états à l'aide d'un réseau de neurones. Nous avons construit un réseau de neurones, qui a pour but d'estimer si la position prise en entrée est bénéfique pour le joueur ou non. Ainsi, la Q-value d'une action est calculé comme la différence entre l'état précédent l'action et l'état après l'action. L'agent choisiras donc l'action qui l'amène dans une position la plus favorable possible.

Pour entraîner cet agent, il faut lui faire jouer un grand nombre de parties et retenir toutes les positions de la partie. A la fin d'une partie, nous optimiserons le réseau de neurones, estimateur de la position, en fonction du gagnant. En effet, si le joueur a gagné, le réseau doit apprendre que les positions qu'il a rencontré lui étaient favorables. Dans le cas où il a perdu, les positions lui étaient défavorables.

2.2. Modulations étudiées :

Nous avons amené deux modulations qui nous semble pertinente :

- Une part d'aléatoire. Suivant une probabilité fixé au début et diminuant avec le temps, notre agent jouera un coup choisi aléatoirement. Cet aspect a pour but d'éviter les minimas locaux et d'explorer plus facilement l'espace.
- Une modulation des récompenses. Afin d'apporter plus d'impacts aux positions de fin de parties, nous avons choisi de calculer les récompenses comme une somme cumulé. Ainsi, si l'agent a gagné la partie, la position en tour 2 a un score de 2 au lieu de 1.

2.3. Réseaux :

D'un point de vue implémentation, nous avons choisi d'utiliser deux réseaux de taille différentes. Le premier volontairement petit au vue du nombre de possibilités et de la taille de la grille (6*7) est composé de :

- Deux couches de convolutions (noyau de tailles 2, filters 2 et muni d'un zéro padding, d'une fonction d'activation Relu ainsi qu'une couche de normalisation par batch)
- Une couche de MaxPooling de taille 2,
- Une couche d'aplatissage,
- Dex couches denses.

Le second volontairement grand pour valider le premier est composé de :

- 8 couches de convolutions (noyau de tailles 2 filters 2,4,8,16,16,8,4,2 et muni d'un zéro padding, d'une fonction d'activation Relu ainsi qu'une couche de normalisation par batch)
- Une couche de MaxPooling de taille 2,
- Une couche d'aplatissage,
- 4 couches denses.

Nous avons choisi de converser l'optimiseur de type Adam dans tous les tests que nous ferons. Ainsi, les deux hyperparamètres que nous étudierons seront le learning rate, la proportion de coups aléatoires ainsi que le nombre d'époques.

3. Résultats

3.1. Influence du learning rate :

Le learning influence fortement la convergence du réseau de neurones. En effet, s'il est supérieur à une valeur seuil, ici 0.001, le réseau diverge prédisant soit uniquement 1 soit uniquement 0. **Utilisation d'une fonction de normalisation Sigmoid sur la dernière couche.**

3.2. Evaluation :

Pour evaluer nos differents agents, j'ai choisi d'effectuer un tournoi, composé de poules de matchs de 25 puis d'un tableau avec des matchs de 51. L'objectif est de pouvoir au mieux comparer les agents.

Nous etudierons donc 9 agents :

- Un agent avec un reseau de taille petite, 1000 epoques d'entrainement et une modulation des recompenses, ie les recompenses sont la somme cumulée.
- Un agent avec un reseau de taille petite, 3000 epoques d'entrainement et une modulation des recompenses.
- Un agent avec un reseau de taille petite, 1000 epoques d'entrainement sans une modulation des recompenses, et avec une fonction d'activation sigmoid sur la derniere couche.
- Un agent avec un reseau de taille petite, 3000 epoques d'entrainement sans une modulation des recompenses, et avec une fonction d'activation sigmoid sur la derniere couche.
- Un agent avec un reseau de taille grande, 1000 epoques d'entrainement et une modulation des recompenses, ie les recompenses sont la somme cumulée.
- Un agent avec un reseau de taille grande, 3000 epoques d'entrainement et une modulation des recompenses.
- Un agent avec un reseau de taille grande, 1000 epoques d'entrainement sans une modulation des recompenses, et avec une fonction d'activation sigmoid sur la derniere couche.
- Un agent avec un reseau de taille grande, 3000 epoques d'entrainement sans une modulation des recompenses, et avec une fonction d'activation sigmoid sur la derniere couche.
- Un agent completement aleatoire.

Ainsi, la poule est la suivante :

	$A_{1000,swm}$	$A_{3000,swm}$	$A_{1000,s}$	$A_{3000,s}$	$A_{1000,lwm}$	$A_{3000,lwm}$	$A_{1000,l}$	$A_{3000,l}$	A_R
$A_{1000,swm}$	[9. 0.]	4	4	5	9	0	0	4	4
$A_{3000,swm}$	5	[5,4]	5	4	4	5	9	5	3
$A_{1000,s}$	5	4	[5,4]	9	4	5	5	5	4
$A_{3000,s}$	4	5	0	[4,5]	4	9	4	9	4
$A_{1000,lwm}$	0	5	5	5	[4,5]	5	5	5	4
$A_{3000,lwm}$	9	4	4	0	4	[5,4]	5	5	4
$A_{1000,l}$	9	0	4	5	4	4	[5,4]	9	0
$A_{3000,l}$	5	4	4	0	4	4	0	[5,4]	4
A_R	5	6	5	5	5	5	4	5	[2. 7.]

Table 1. Résultats des poules

On constate qu'il n'y a pas de véritables vainqueurs. De plus, les agents ne battent pas souvent l'agent jouant de manière aléatoire. On peut donc voir qu'ils ne sont pas forcément très performant.

4. Résolutions de problèmes plus complexes :

On peut résumer les opportunités de nos agents dans le tableau suivant :

Method	backgammon	Chess	Go	Starcraft
State Evaluation	x	x	x	.
DeepQ learning Evaluation	.	.	.	.

Table 2. opportunités à résoudre des problèmes plus complexes

Cet agent a pour avantage d'être très générique. Il ne nécessite pas de spécificité à part le fait que l'entrée doit être mesurable. Il est donc clairement possible de l'utiliser pour des jeux du type backgammon, Chess ou Go. On peut déjà cependant noter que ces résultats seront faibles, mais si on ajoute un aspect de recherche dans un arbre de meilleur situation, on retombe sur des algorithmes comme AlphaGo. Notre agent effectue un réduction de ces approches, en utilisant notamment le fait que les actions soient très limités dans notre problème. Cet aspect peut poser des problèmes de performances, et des soucis de calculs. Mais notre approche est tout de même utilisable. On peut notamment citer l'article suivant [3], qui implémenté une idée similaire avec une feature augmentation pour les échecs.

Pour Starcraft, le problème est plus complexe qu'un simple jeu en temps réel. En effet, dans Starcraft, les informations ne sont ni une grille ni une image. Elles sont mixtes entre images (cartes et mini-cartes) et informations macroscopiques (ressources stratégiques). Ainsi, une méthode similaire peut être envisagé. AlphaStar se base également sur des réseaux de neurones [4], mais il est selon nous impossible de comparer notre approche à celle réalisé par AlphaStar de part la simplicité de nos réseaux, de nos entrées et notamment du nombre d'actions restreint de notre problème. De plus, AlphaStar fonctionne avec deux réseaux, un macroscopique définissant l'action, un microscopique effectuant l'action de la meilleure manière possible. Notre agent n'est clairement pas capable de cette prouesse.

5. Conclusion

En conclusion, on peut noter que les agents ne sont pas très forts. En jouant quelques parties contre eux, on peut s'apercevoir qu'ils possèdent un schéma de coup qu'il semble lui permettre de gagner.

Pour améliorer notre agent, on pourrait pré-entraîner notre agent sur des bases de données de parties ou le laisser jouer beaucoup plus longtemps. Dans la littérature, j'ai trouvé des articles et des études, disant que cette méthode n'aboutit pas souvent à des agents performants. AlphaGo est certes basé dessus mais la recherche dans l'arbre est probablement la brique qui permet de passer d'un agent moyen à un bonne état. Mais il ne faut pas négliger la différence de calculs énormes entre mon étude et les serveurs de DeepMind.

Ainsi, construire un agent basé sur du Reinforcement Learning est possible mais construire un agent performant est nettement plus compliqué. Dans notre groupe, nous n'avons pas réussi à construire d'agents performants.

References

[1] Wikipédia, "Puissance 4," *Wikipédia*,

[2] DeepMind, "Alphago,"

[3] M. Lai, "Giraffe: Using deep reinforcement learning to play chess," *CoRR*, vol. abs/1509.01549, 2015. arXiv: 1509.01549. [Online]. Available: http://arxiv.org/abs/1509.01549.

[4] DeepMind, "Alphastar,"